国际化官网项目设计

背景

C端根据调研要进行品牌名变更（英语母语国家为MyTopia，非英语母语国家为Fizzo），因原创作者国家分布主要集中在印尼 36%，菲律宾 21%，尼日利亚21%，以上国家C端上架产品为Fizzo，为保证大部分作者对平台认知的一致性，作家平台改为Fizzo。为了增加作者对品牌的认知和信任度，官网需要从面向C端用户导流APP，更改为面向作者进行原创品牌展示和引导入驻。同时在官网完全改版的基础上进行SEO相关的优化。

总体设计

技术选型

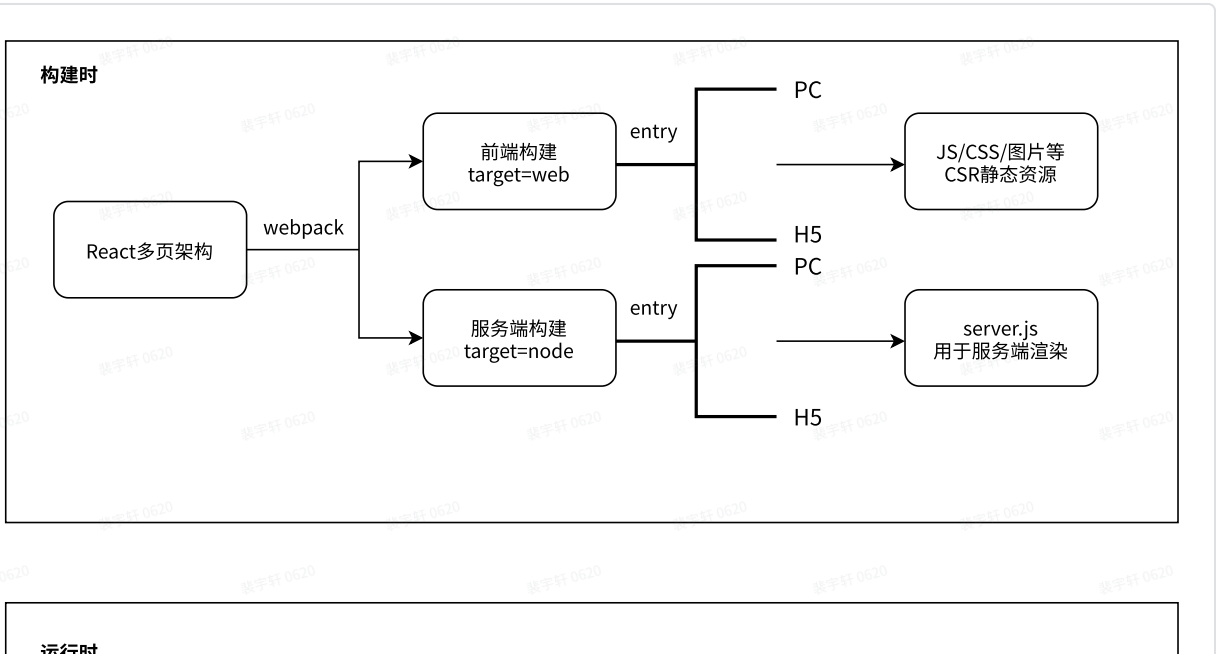
整体以React技术栈为核心，采用**React SSR + Node BFF**同构方案实现：

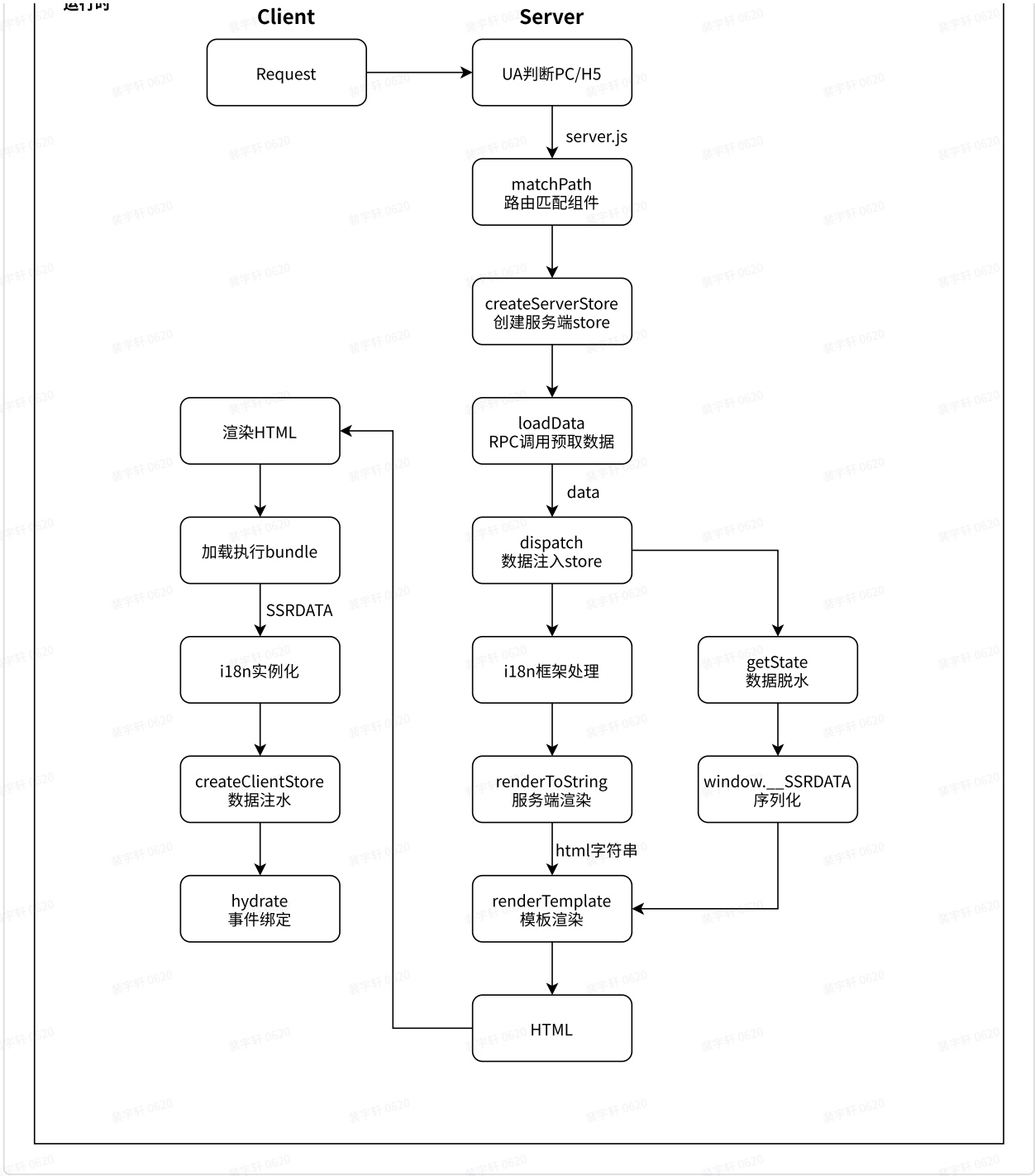
- 前端：React + Redux + Typescript
- 服务端：Gulu + Goofy Node
- i18n：starling cli + intl框架 + intl-react插件

架构设计

国际化官网需要兼顾PC端和移动端设备，且页面UI及交互差异性较大，因此项目整体采用React多页面架构，PC和H5页面入口独立：

- 构建时：分别进行前端与Node产物构建
- 运行时：根据UA判定设备类型，获取对应的服务端构建产物进行页面渲染处理





文案处理

服务端渲染页面后，客户端会直接复用DOM，仅进行交互事件的绑定。因此SSR场景下做国际化文案处理时需要保证服务端与客户端语种文案的一致性，否则就会出现**文案前后翻译不一致错误**

实现方案

- localstorage

作家后台目前采用localstorage本地存储当前语言设置，有登录态时通过接口做持久化处理。但SSR场景下无法读取，当前方案不可行

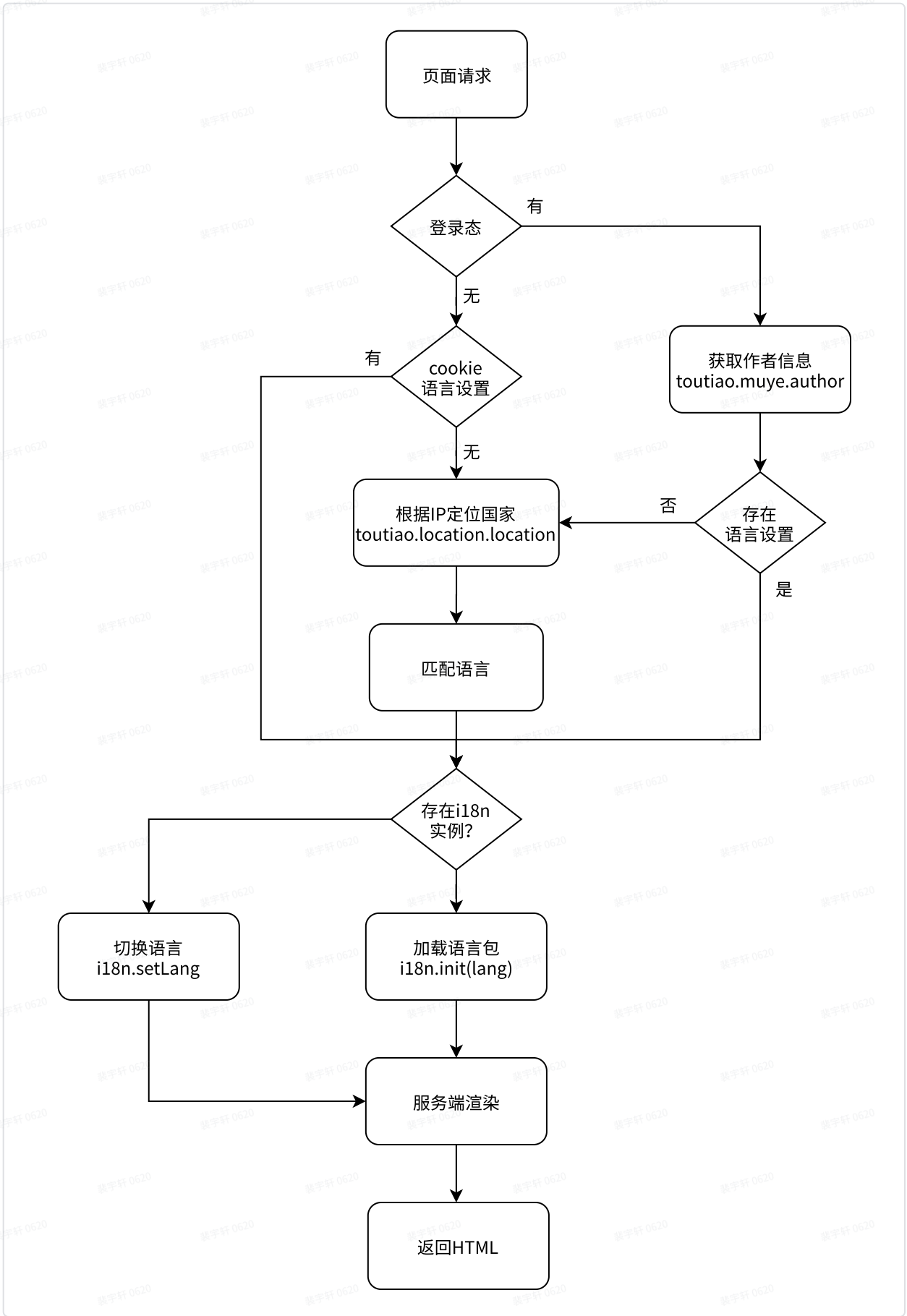
- Cookie

通过本地操作Cookie写入当前语种，在服务端处理请求时可以从header中读取字段设置，客户端则通过document.cookie属性获取语言设置。从而实现前后端渲染的文案一致性

实现流程

服务端

1. 采用单例模式：考虑性能，服务运行时只初始化一个i18n实例，后续请求通过setLang切换语言
2. 语言优先级处理：
    a. 已登录用户存在自定义语言则直接使用
    b. Cookie中存在语言设置则直接使用
    c. 获取用户请求的真实IP，接入地理中台服务来根据IP定位用户所在国家，然后匹配语言
    d. 以上逻辑发生异常时，使用英语作为兜底语言
3. 确定当前页面请求语言后，在返回客户端时将语言挂载到全局变量



客户端

1. 根据服务端渲染时挂载的全局变量来初始化客户端i18n实例
2. 用户切换语言时手动写入Cookie设置

C++

```
I18n.setLang(lang, () => {
    Cookies.set(LANG_KEY, lang);
})
```

SSR降级

在进行服务端渲染时，可能出现渲染、接口等异常，需要对各种异常情况进行处理，为了确保用户请求能够正常响应，必要时降级CSR方案：

1. 接口异常：服务端loaddata失败后，仅进行基本页面内容渲染，客户端渲染时重新获取数据
2. 渲染异常：返回CSR页面模板，浏览器端加载JS资源后进行客户端渲染
3. 超时处理：设置服务端渲染总时长（接口+模板渲染）阈值，超时后降级为CSR渲染

核心逻辑

动画实现

Scroll-driven 滚动驱动式动画实现

图片加载优化

首页包含大量图片资源，同时并发加载时受用户网络带宽等限制，请求开销过大会导致页面渲染耗时过长，用户体验较差。且不在用户视野范围内的图片可能实际并未真正使用到。

1. **图片懒加载**

主要针对第二屏及后面的图片加载，通过给img标签设置自定义属性data-src存放真正的图片地址，待元素进入浏览器视口时再将地址赋值给src属性，开始加载：

a. IntersectionObserver：创建observer实例监听目标元素是否进入可视范围
b. 设置src属性，将目标元素从observer实例的观察列表中移除

JavaScript

```
const list = Array.from(el.querySelectorAll('.lazy-load') || []);
let observer = new IntersectionObserver(entries => {
    entries.forEach(entry => {
        // 进入可视范围的百分比
        if (entry.intersectionRatio > 0) {
            const imgList = Array.from(entry.target.querySelectorAll('img'));
            imgList.forEach(img => {
            if (img.dataset.src) {
                img.src = img.dataset.src;
             }
         });
         observer.unobserve(entry.target);
       }
    });
});
list.forEach(item => observer.observe(item));
```

2. **渐进式加载**

针对第一屏多张高分辨率图的加载过慢问题，采用低分辨到高分辨渐进性加载方式进行优化：

a. 首先快速加载低分辨率版本的图片

b. 原图通过隐藏div背景并行加载

c. 页面渲染完成后，通过缓存加载原图触发onload回调，在回调中处理替换低分辨率图

JavaScript

```
// 大图再加载
    useEffect(() => {
        loadImgAsync(bgImg).then(() => setBgLoaded(true));
        loadImgAsync(bookImg).then(() => setBookLoaded(true));
        loadImgAsync(ribbonImg).then(() => setRibbonLoaded(true));
    }, []);
```

3.

监控与报警

服务运维

1. @byted-service/metrics：QPS、接口时延等核心指标自动上报
2. @byted-service/logger：关键业务日志上报

前端监控

1. Slardar SDK：页面性能数据与JS异常监控上报
2. Mue SDK：原创业务通用自定义异常打点上报

上线方案

国际化官网上线方案